REMARKS

Applicant respectfully submits that entry of the currently amended claims above is proper because the currently amended claims will place the application in condition for allowance or in better form for appeal.

Claims 1, 5-8, 11-13, 17-19, 23, and 24 are pending in the application. Claims 1, 5-8, 11-13, 17-19, 23, and 24 are currently amended. Claim 25 is canceled. No new matter is added to currently amended claims 1, 5-8, 11-13, 17-19, 23, and 24.

Claims 19 and 23-25 stand rejected under 35 U.S.C. §101.

Claims 1, 5-8, 11-13, 17-19 and 23-25 stand rejected under 35 U.S.C. §102(b) as anticipated by U.S. Patent No. 6,144,954 to Li.

Applicant respectfully traverses these rejections based on the following discussion

I. The 35 U.S.C. §101 Rejection

The Office Action rejects claims 19 and 23-25 under 35 U.S.C. §101 because the claim limitations are directed towards computer programs *per se*. The Office Action further asserts that computer programs constitute non-statutory subject matter under the statute.

Applicant respectfully submits that independent claim 19 is currently amended to recite in relevant parts, "A computer system comprising: a pre-compiler …; and a processor … ." Applicant further respectfully submits that these amendments fulfill the statutory requirements of 35 U.S.C. §101 because the computer system, the pre-compiler, and the processor are structures.

Claims 23 and 24 depend from currently amended, independent claim 19. The rejection of canceled claim 25 is moot.

In view of the foregoing, the Examiner is respectfully requested to reconsider and withdraw the rejection of claims 19 and 23-25 under 35 U.S.C. §101.

II. The 35 U.S.C. §102(b) Rejection over Li

A. The Li Reference

Li discloses that the pre-specified optimizing criteria for a computer software generation, modification, use, maintenance, or performing a specific function may be a feature selected from a group consisting of performance characteristics, power, portability, cost of development, productivity, smallness, maintainability, flexibility, memory requirement, output quality, and computing or analyzing speed. (emphasis added) (col. 21, lines 16-23, which are cited by the Office Action at: page 3, printed line 7; page 5, printed line 4; page 7, printed line 6; and page 8, printed lines 4 and 5; in reference to independent claims 1, 7, 13, and 19, respectively).

B. The Claimed Invention

Claims 1 and 13 recite in relevant part,

"inputting a selection command that selects one function from a list of pre-selected functions for input into said computer program at a point of choice determined by a programmer, wherein each function from said list of pre-selected functions is associated with a reward".

Similarly, claim 7 recites in relevant part,

"inputting a selection command that selects one function from a list of pre-selected functions for input into said computer program, wherein each function from said list of pre-selected functions is associated with a reward".

Similarly, claim 19 recites in relevant part,

"a pre-complier that inputs a selection command at a point of choice, determined by a programmer, into a computer program that runs on a computer, said selection command selecting one function from a list of pre-selected functions for input into said computer program, wherein each function from said list of pre-selected functions is associated with a reward".

C. Arguments

Li discloses optimizing criteria for computer software generation, etc., that may utilize a feature selected from a group consisting of performance characteristics, power, portability, cost of development, productivity, smallness, maintainability, flexibility, memory requirement, output

quality, and computing or analyzing speed. Li's features, to one of ordinary skill in the art, are recognized as variables or variable inputs, such as, for example, a desired power level, which acts as an optimal criterion.

In contrast, the present invention "selects one function from a list of pre-selected functions for input into said computer program". The Specification, at paragraph **[0019]**, describes an exemplary function as "various sorting algorithms". A function, as defined by the 4th edition of the Microsoft Computer Dictionary, and as known to one of ordinary skill in the art, may be considered as a subroutine that may return a value dependent on its inputs. A function is not a variable. Applicant respectfully submits that the variables of Li are improperly analogized to the functions of the present invention.

For at least the reasons outlined above, Applicant respectfully submits that Li does not disclose, teach or suggest the present invention's claimed features of: "inputting a selection command that selects one function from a list of pre-selected functions for input into said computer program at a point of choice determined by a programmer, wherein each function from said list of pre-selected functions is associated with a reward", for claims 1 and 13; "inputting a selection command that selects one function from a list of pre-selected functions for input into said computer program, wherein each function from said list of pre-selected functions is associated with a reward", for claim 7; and "a pre-complier that inputs a selection command at a point of choice, determined by a programmer, into a computer program that runs on a computer, said selection command selecting one function from a list of pre-selected functions for input into said computer program, wherein each function from said list of pre-selected functions is associated with a reward", for claim 19. Accordingly, Li fails to anticipate the subject matter of currently amended, independent claims 1, 7, 13, and 19, and currently amended, dependent claims 5, 6, 8, 11, 12, 17, 18, 23, and 24 under 35 U.S.C. §102(b). Therefore, Li does not anticipate the claimed invention under 35 U.S.C. §102(b). The rejection of canceled claim 25 is moot. Withdrawal of the rejection of claims 1, 5-8, 11-13, 17-19, and 23-25 is respectfully solicited.

D. Additional Arguments

Furthermore, Li teaches finding the best rule from a *pre-selected* set of possible rules. In contrast, the present invention teaches how the program automatically *builds its own rules*, whereas the software engineer merely suggests (by encoding the suggestion formally within the code) which program functions may be informative for the program to rely upon while it creates its rules tailored to particular environments in which it may run.

Further, Li's "Summary of the Invention" discloses the narrow scope of its underlying methodology (which may have many applications though, but not for self-optimization of software),

> "[P]lanning an efficient, statistical design matrix of n tests on m variables ...; performing the necessary tests; collecting and recording the sets of data on the critical performances to be optimized; analyzing the collected data to determine the functional relationships between the m variables and the critical performances and to obtain the instantaneous optimal variable combinations; setting the m variables to the thus-computed optimal combinations before these combinations change; and feeding back information on the status of optimization to achieve a closed-loop feed-back control. The knowledge bases so generated are instantly machine-coded. a machine operating on the basis results of this method is also disclosed."

Further evidence of this rather narrow-scope methodology of a statistical design matrix appears in col. 6, lines 19 and 20:

> "The computer-planned fractional factor experiment makes only 8 tests."

Instead, Li describes several "embodiments: all of which are examples where software can be developed for optimizing some systems. Li fails to teach a general method for building software that optimizes its own performance other than what is described in the Summary of Invention. Therefore, Li does not anticipate the present invention. Note that charts in Li describe particular embodiments rather than a general method.

Further, the teaching of Li in regard to automatic development of software amounts to the one paragraph description of Fig. 4,

"Fig. 4 shows the procedural steps for the disclosed self-optimization method and machine used, for example, in the automatic development of computer software. As shown in Fig. 4, the self-optimizing method includes the steps of receiving an initial knowledge set in step 78 to be associated with the computer software in step 80; instantly computer-coding the computer-generated knowledge base in step 82 to generate the computer software storage device such as a memory in step 86; and using the automatically developed computer software under control of the automation manager in step 88 to generate associated outputs of the automatically developed computer software. The computer-generating, computer-coding, and saving steps may all be performed in real-time so that the computer software is developed in real time."

The above description by Li merely repeats what is already written in Fig. 4. The above paragraph does not teach the building of a self-optimizing program. All the paragraph dislcoses is "receive knowledge set," "generate knowledge base," "code the knowledge base," "store the code," and so on. There is no teaching of how to create software that learns an optimal strategy of making decisions at decision points and how to cause this strategy to adapt to a changing environment as in the present invention.

Further, Li's Fig. 2, relates to a furnace to make special iron-based alloys (col. 6, lines 1-2). the furnace has a built-in microprocessor that "plans a statistical design matrix of operating or alloying conditions" ... Specifically, 8 tests or allotting conditions are planned." (col. 6, lines 13-17). Thus, the processor is programmed to optimize the furnace – it does not self-optimize its own software. This limitation is highlighted again in col. 6, lines 54-64, where Li mentions more possibilities of "self-optimizing machines" where in each case a microprocessor seeks to optimize a machine but does not optimize its own software.

None of the embodiments of Li teach or suggest a method that a software engineer can use in the design of software in general that *optimizes its own performance*.

This distinction is also manifested in the description of Fig. 3 "The sensors and actuators used, however, are generally different from those used in the self-optimizing furnace" (col. 9, lines 9-11) – again the processor optimizes a machine rather than its own performance.

Once again, in the embodiment suggested in Fig. 3 from col. 9, line 53 to col. 10, line 36, the software is designed to optimize something else rather than its own performance.

With regard to Fig. 6, again the optimization is with respect to data from a "plurality of sensors … to optimally control a plurality of actuators." (col. 11, lines 14-15). Again, a program optimizes another system rather than its own performance.

Yet again, in col. 12, line 20-25, Li describes "Multiple sensors sense numerical data … ." The software is designed to search for drugs. It does not optimize itself. Applicant respectfully submits that software for solving optimization problems, even in real time, existed long ago.

Context for an exemplary embodiment of the present invention

Applicant respectfully submits that the software engineer does not know in advance the particulars of all the environments in which the code may be run. In particular, the engineer cannot tell in advance which data structure would be optimal to support some parts of the code in various changing environments, and which algorithms would perform better. The correct answer almost always is "it depends." It is the dependency itself that the self-optimizing code of the present invention is programmed to figure out and then make the best choices with respect to the current conditions. in the present invention, the programmer does not have to code the learning algorithm. The programmer simply lists the factors that should go into the decision and refers to a particular learning algorithm from a library of learning algorithms that would be available to many other programmers, and the code would search for the optimal function using tools from the library. The result could be that if the program values V1,…Vn currently satisfy the condition that $2V1+V2 < V3*V4$ then the software should take a path that uses a certain search algorithm and a certain data structure. The programmer plants such optimization tasks throughout the code.

III. Formal Matters and Conclusion

Claims 1, 5-8, 11-13, 17-19, 23, and 24 are pending in the application.

Applicants respectfully submit that entry of the currently amended claims above is proper because the currently amended claims will place the application in condition for allowance or in better form for appeal. Applicants further respectfully submit that no new matter is added to the currently amended claims.

With respect to the rejections to the claims, the claims have been amended, above, to overcome these rejections. In view of the foregoing, the Examiner is respectfully requested to reconsider and withdraw the rejections to the claims.

In view of the foregoing, Applicants submit that claims 1, 5-8, 11-13, 17-19, 23, and 24, all the claims presently pending in the application, are patentably distinct over the prior art of record and are in condition for allowance. The Examiner is respectfully requested to pass the above application to issue at the earliest possible time.

Should the Examiner find the application to be other than in condition for allowance, the Examiner is requested to contact the undersigned at the local telephone number listed below to discuss any other changes deemed necessary. Please charge any deficiencies and credit any overpayments to Attorney's Deposit Account Number 09-0441.

Respectfully submitted,

Dated: December 14, 2007

/Peter A. Balnave/
Peter A. Balnave, Ph.D.
Registration No. 46,199

Gibb & Rahman, LLC
2568-A Riva Road, Suite 304
Annapolis, MD 21401
Voice: (410) 573-5255
Fax: (301) 261-8825
Customer Number: 29154